Appendix A:

This appendix contains the complete source code to two algorithm modules that exemplify an embodiment of the current invention: a finite impulse response filter module (FIR) and a filter group module (FIG). Although a digital filter is much too simple an algorithm to encapsulate as a component, it illustrates (and hopefully motivates) the concepts presented in the specification. The FIR filter example consists of the following files:

1) fir.c, fir.h – FIR utility API module source and interface header
2) ifir.c, ifir.h – abstract FIR interface definition header and parameter defaults
3) fir_ti.c, fir_ti.h – vendor specific implementation and header
4) fir_ti_ext.c – vendor specific extensions to FIR
5) firtest.c, firtest1.c – simple programs using ALG to execute a FIR filter.

The filter group module, FIG, is an example that illustrates how multiple instances of an algorithm can be grouped together to share common coefficients.

The filter group example consists of the following files.

1) fig.c, fig.h – FIG utility API module source and interface header
2) ifig.h – abstract FIG interface definition header
3) fig_ti.c, fig_ti.h – vendor specific implementation and header
4) figtest.c – a simple program using ALG to execute a filter group.

Table A-1 summarizes a characterization of the performance of the FIR example, including memory usage requirements. A similar characterization can be compiled for the FIG example.

Instance Parameters	
filterlen	16
framelen	180

Other Parameters	
word size (bytes)	2
sample rate (samp/sec)	8000

Execution Time	Period	Cycles/Period
worst case	22500 us	2880

Interrupt Latency	0 cycles

Stack Memory	Size	Align
worst case	40	0

Instance Memory	DARAM		SARAM		External	
	Size	Align	Size	Align	Size	Align
scratch	390	0	0	0	0	0
persistent	0	0	0	0	42	0

Module Memory	Code		Data		BSS	
	Size	Align	Size	Align	Size	Align
fir_ti.o54	734	0	0	0	34	0
fir_ti_ext.o54	134	0	0	0	0	0
fir_ti_irtc_o54	58	0	0	0	6	0

TABLE A-1

Name **fir.h – FIR Module Interface**

Text

```
/*
 *  ======== fir.h ========
 *  This header defines all types, constants, and functions used by
 *  applications that use the FIR algorithm.
 *
 *  Applications that use this interface enjoy type safety and
 *  the ability to incorporate multiple implementations of the FIR
 *  algorithm in a single application at the expense of some
 *  additional indirection.
 */

#ifndef FIR_
#define FIR_

#include <alg.h>
#include <ifir.h>
#include <ialg.h>


/*
 *  ======== FIR_Handle ========
 *  FIR algorithm instance handle
 */
typedef struct IFIR_Obj *FIR_Handle;


/*
 *  ======== FIR_Params ========
 *  FIR algorithm instance creation parameters
 */
typedef struct IFIR_Params FIR_Params;


/*
 *  ======== FIR_PARAMS ========
 *  Default instance parameters
 */
#define FIR_PARAMS IFIR_PARAMS


/*
 *  ======== FIR_apply ========
 *  Apply a FIR filter to the input array and place results in the
 *  output array.
 */
extern Void FIR_apply(FIR_Handle fir, Int in[], Int out[]);
```

```
/*
 *  ======== FIR_create ========
 *  Create an instance of a FIR object.
 */
static inline FIR_Handle FIR_create(const IFIR_Fxns *fxns,
                                    const FIR_Params *prms)
{
    return ((FIR_Handle)ALG_create((IALG_Fxns *)fxns,
                                    NULL, (IALG_Params *)prms));
}

/*
 *  ======== FIR_delete ========
 *  Delete a FIR instance object
 */
static inline Void FIR_delete(FIR_Handle handle)
{
    ALG_delete((ALG_Handle)handle);
}

/*
 *  ======== FIR_exit ========
 *  Module finalization
 */
extern Void FIR_exit(Void);

/*
 *  ======== FIR_init ========
 *  Module initialization
 */
extern Void FIR_init(Void);

#endif  /* FIR_ */
```

Name **ifir.h – Example Abstract FIR Filter Interface**

Text

```
/*
 *  ======== ifir.h ========
 *  This header defines all types, constants, and functions shared by all
 *  implementations of the FIR algorithm.
 */
#ifndef IFIR_
#define IFIR_

#include <ialg.h>

/*
 *  ======== IFIR_Obj ========
 *  Every implementation of IFIR *must* declare this structure as
 *  the first member of the implementation's object.
 */
typedef struct IFIR_Obj {
    struct IFIR_Fxns *fxns;
} IFIR_Obj;

/*
 *  ======== IFIR_Handle ========
 *  This type is a pointer to an implementation's instance object.
 */
typedef struct IFIR_Obj *IFIR_Handle;

/*
 *  ======== IFIR_Params ========
 *  This structure defines the parameters necessary to create an
 *  instance of a FIR object.
 *
 *  Every implementation of IFIR *must* declare this structure as
 *  the first member of the implementation's parameter structure.
 */
typedef struct IFIR_Params {
    Int size;           /* sizeof the whole parameter struct */
    Int *coeffPtr;      /* pointer to coefficients */
    Int filterLen;      /* length of filter */
    Int frameLen;       /* length of input (output) buffer */
} IFIR_Params;

/*
 *  ======== IFIR_PARAMS ========
 *  Default instance creation parameters (defined in ifir.c)
 */
extern IFIR_Params IFIR_PARAMS;
```

```
/*
 *  ======== IFIR_Fxns ========
 *  All implementation's of FIR must declare and statically
 *  initialize a constant variable of this type.
 *
 *  By convention the name of the variable is FIR_<vendor>_IFIR, where
 *  <vendor> is the vendor name.
 */
typedef struct IFIR_Fxns {
    IALG_Fxns   ialg;
    Void        (*filter)(IFIR_Handle handle, Int in[], Int out[]);
} IFIR_Fxns;

#endif  /* IFIR_ */
```

Name **fir.c – Common FIR Module Implementation**

Text

```
/*
 *  ======== fir.c ========
 *  FIR Filter Module - implements all functions and defines all constant
 *  structures common to all FIR filter algorithm implementations.
 */
#include <std.h>
#include <alg.h>

#include <fir.h>

/*
 *  ======== FIR_apply ========
 *  Apply a FIR filter to the input array and place results in the
 *  output array.
 */
Void FIR_apply(FIR_Handle handle, Int in[], Int out[])
{
    /* activate instance object */
    ALG_activate((ALG_Handle)handle);

    handle->fxns->filter(handle, in, out);      /* filter data */

    /* deactivate instance object */
    ALG_deactivate((ALG_Handle)handle);
}

/*
 *  ======== FIR_exit ========
 *  Module finalization
 */
Void FIR_exit()
{
}

/*
 *  ======== FIR_init ========
 *  Module initialization
 */
Void FIR_init()
{
}
```

Name

fir_ti.c – Vender Specific FIR Module Implementation

Text

```
/*
 *  ======== fir_ti_ialg.c ========
 *  FIR Filter Module - TI implementation of a FIR filter algorithm
 *
 *  This file contains an implementation of the IALG interface
 *  required by XDAIS.
 */
#pragma CODE_SECTION(FIR_TI_activate, ".text:algActivate")
#pragma CODE_SECTION(FIR_TI_alloc, ".text:algAlloc()")
#pragma CODE_SECTION(FIR_TI_deactivate, ".text:algDeactivate")
#pragma CODE_SECTION(FIR_TI_free, ".text:algFree")
#pragma CODE_SECTION(FIR_TI_initObj, ".text:algInit")
#pragma CODE_SECTION(FIR_TI_moved, ".text:algMoved")

#include <std.h>

#include <ialg.h>
#include <ifir.h>
#include <fir_ti.h>
#include <fir_ti_priv.h>

#include <string.h>          /* memcpy() declaration */

#define HISTORY 1
#define WORKBUF 2
#define NUMBUFS 3

/*
 *  ======== dot ========
 */
static Int dot(Int *a, Int *b, Int n)
{
    Int sum = 0;
    Int i;

    for (i = 0; i < n; i++) {
        sum += *a++ * *b++;
    }
    return (sum);
}
```

```
/*
 *  ======== FIR_TI_activate ========
 *  Copy filter history from external slow memory into working buffer.
 */
Void FIR_TI_activate(IALG_Handle handle)
{
    FIR_TI_Obj *fir = (Void *)handle;

    /* copy saved history to working buffer */
    memcpy((Void *)fir->workBuf, (Void *)fir->history,
        fir->filterLenM1 * sizeof(Int));
}
```

```
/*
 *  ======== FIR_TI_alloc ========
 */
Int FIR_TI_alloc(const IALG_Params *algParams,
                 IALG_Fxns **pf, IALG_MemRec memTab[])
{
    const IFIR_Params *params = (Void *)algParams;

    if (params == NULL) {
        params = &IFIR_PARAMS;  /* set default parameters */
    }

    /* Request memory for FIR object */
    memTab[0].size = sizeof(FIR_TI_Obj);
    memTab[0].alignment = 0;
    memTab[0].space = IALG_EXTERNAL;
    memTab[0].attrs = IALG_PERSIST;

    /*
     *  Request memory filter's "inter-frame" state (i.e., the
     *  delay history)
     *
     *  Note we could have simply added the delay buffer size to the
     *  end of the FIR object by combining this request with the one
     *  above, thereby saving some code.  We separate it here for
     *  clarity.
     */
    memTab[HISTORY].size = (params->filterLen - 1) * sizeof(Int);
    memTab[HISTORY].alignment = 0;
    memTab[HISTORY].space = IALG_EXTERNAL;
    memTab[HISTORY].attrs = IALG_PERSIST;

    /*
     *  Request memory for shared working buffer
     */
    memTab[WORKBUF].size =
        (params->filterLen - 1 + params->frameLen) * sizeof(Int);
    memTab[WORKBUF].alignment = 0;
    memTab[WORKBUF].space = IALG_DARAM0;
    memTab[WORKBUF].attrs = IALG_SCRATCH;

    return (NUMBUFS);
}
```

```
/*
 *  ======== FIR_TI_deactivate ========
 *  Copy filter history from working buffer to external memory
 */
Void FIR_TI_deactivate(IALG_Handle handle)
{
    FIR_TI_Obj *fir = (Void *)handle;

    /* copy history to external history buffer */
    memcpy((Void *)fir->history, (Void *)fir->workBuf,
        fir->filterLenM1 * sizeof(Int));
}

/*
 *  ======== FIR_TI_filter ========
 */
Void FIR_TI_filter(IFIR_Handle handle, Int in[], Int out[])
{
    FIR_TI_Obj *fir = (Void *)handle;
    Int *src = fir->workBuf;
    Int *dst = out;
    Int i;

    /* copy input buffer into working buffer */
    memcpy((Void *)(fir->workBuf + fir->filterLenM1), (Void *)in,
        fir->frameLen * sizeof (Int));

    /* filter data */
    for (i = 0; i < fir->frameLen; i++) {
        *dst++ = dot(src++, fir->coeff, fir->filterLenM1 + 1);
    }

    /* shift filter history to start of work buffer for next frame */
    memcpy((Void *)fir->workBuf, (Void *)(fir->workBuf + fir->frameLen),
        fir->filterLenM1 * sizeof (Int));
}
```

```
/*
 *  ======== FIR_TI_free ========
 */
Int FIR_TI_free(IALG_Handle handle, IALG_MemRec memTab[])
{
    FIR_TI_Obj *fir = (Void *)handle;

    FIR_TI_alloc(NULL, NULL, memTab);

    memTab[HISTORY].base = fir->history;
    memTab[HISTORY].size = fir->filterLenM1 * sizeof(Int);

    memTab[WORKBUF].size =
        (fir->filterLenM1 + fir->frameLen) * sizeof(Int);
    memTab[WORKBUF].base = fir->workBuf;

    return (NUMBUFS);
}

/*
 *  ======== FIR_TI_initObj ========
 */
Int FIR_TI_initObj(IALG_Handle handle,
                const IALG_MemRec memTab[], IALG_Handle p,
                const IALG_Params *algParams)
{
    FIR_TI_Obj *fir = (Void *)handle;
    const IFIR_Params *params = (Void *)algParams;

    if (params == NULL) {
        params = &IFIR_PARAMS;  /* set default parameters */
    }

    fir->coeff = params->coeffPtr;
    fir->workBuf = memTab[WORKBUF].base;
    fir->history = memTab[HISTORY].base;
    fir->filterLenM1 = params->filterLen - 1;
    fir->frameLen = params->frameLen;

    return (IALG_EOK);
}
```

```
/*
 *  ======== FIR_TI_moved ========
 */
Void FIR_TI_moved(IALG_Handle handle,
                  const IALG_MemRec memTab[], IALG_Handle p,
                  const IALG_Params *algParams)
{
    FIR_TI_Obj *fir = (Void *)handle;
    const IFIR_Params *params = (Void *)algParams;

    if (params != NULL) {
        fir->coeff = params->coeffPtr;
    }

    fir->workBuf = memTab[WORKBUF].base;
    fir->history = memTab[HISTORY].base;
}
```

Name

fir_ti.h – Vender Specific FIR Module Interface

Text

```
/*
 *  ======== fir_ti.h ========
 *  Vendor specific (TI) interface header for FIR algorithm.
 *
 *  Applications that use this interface enjoy type safety and
 *  and minimal overhead at the expense of being tied to a
 *  particular FIR implementation.
 *
 *  This header only contains declarations that are specific
 *  to this implementation.  Thus, applications that do not
 *  want to be tied to a particular implementation should never
 *  include this header (i.e., it should never directly
 *  reference anything defined in this header.)
 */
#ifndef FIR_TI_
#define FIR_TI_

#include <ialg.h>
#include <irtc.h>
#include <itst.h>
#include <ifir.h>

/*
 *  ======== FIR_TI_exit ========
 *  Required module finalization function
 */
extern Void FIR_TI_exit(Void);

/*
 *  ======== FIR_TI_init ========
 *  Required module initialization function
 */
extern Void FIR_TI_init(Void);

/*
 *  ======== FIR_TI_IALG ========
 *  TI's implementation of FIR's IALG interface
 */
extern IALG_Fxns FIR_TI_IALG;

/*
 *  ======== FIR_TI_IFIR ========
 *  TI's implementation of FIR's IFIR interface
 */
extern IFIR_Fxns FIR_TI_IFIR;
```

```
/*
 *  ======== FIR_TI_IRTC ========
 *  TI's implementation of FIR's IRTC interface
 */
extern IRTC_Fxns FIR_TI_IRTC;

/*
 *  ======== Vendor specific methods  ========
 *  The remainder of this file illustrates how a vendor can
 *  extend an interface with custom operations.
 *
 *  The operations below simply provide a type safe interface
 *  for the creation, deletion, and application of TI's FIR filters.
 *  However, other implementation specific operations can also
 *  be added.
 */

/*
 *  ======== FIR_TI_Handle ========
 */
typedef struct FIR_TI_Obj *FIR_TI_Handle;

/*
 *  ======== FIR_TI_Params ========
 *  We don't add any new parameters to the standard ones defined
 *  by IFIR.
 */
typedef IFIR_Params FIR_TI_Params;

/*
 *  ======== FIR_TI_PARAMS ========
 *  Define our defult parameters.
 */
#define FIR_TI_PARAMS   IFIR_PARAMS

/*
 *  ======== FIR_TI_create ========
 *  Create a FIR_TI instance object.
 */
extern FIR_TI_Handle FIR_TI_create(const FIR_TI_Params *params);

/*
 *  ======== FIR_TI_delete ========
 *  Delete a FIR_TI instance object.
 */
extern Void FIR_TI_delete(FIR_TI_Handle handle);
```

```
/*
 *  ======== FIR_TI_nApply ========
 *  Apply specified FIR filter to n input frames and overwrite
 *  input with the result.
 */
extern Void FIR_TI_nApply(FIR_TI_Handle handle, Int inout[], Int n);

#endif  /* FIR_TI_ */
```

Name

fir_ti_priv.h – Private Vender Specific FIR Header

Text

```
/*
 *  ======== fir_ti_priv.h ========
 *  Internal vendor specific (TI) interface header for FIR
 *  algorithm. Only the implementation source files include
 *  this header; this header is not shipped as part of the
 *  algorithm.
 *
 *  This header contains declarations that are specific to
 *  this implementation and which do not need to be exposed
 *  in order for an application to use the FIR algorithm.
 */
#ifndef FIR_TI_PRIV_
#define FIR_TI_PRIV_

#include <ialg.h>
#include <irtc.h>
#include <itst.h>
#include <ifir.h>
#include <log.h>

typedef struct FIR_TI_Obj {
    IALG_Obj    alg;            /* MUST be first field of XDAIS algs */
    IRTC_Mask   mask;           /* current test/diag mask setting */
    Int         *workBuf;       /* on-chip scratch history */
    Int         *coeff;         /* on-chip persistant coeff */
    Int         *history;       /* off chip persistant history */
    Int         filterLenM1;    /* length of coefficient array - 1 */
    Int         frameLen;       /* length of input (output) buffer */
} FIR_TI_Obj;

extern LOG_Obj *FIR_TI_rtcOut;   /* our output trace log */
```

```
/*
 *  ======== FIR_TI_trace ========
 *  Our equivalent of "printf"
 */
 #define FIR_TI_trace(f, a1, a2)                          \
    if (FIR_TI_rtcOut != NULL) {                          \
        LOG_printf(FIR_TI_rtcOut, (f), (a1), (a2)); \
    }

extern Void FIR_TI_activate(IALG_Handle handle);

extern Void FIR_TI_deactivate(IALG_Handle handle);

extern Int FIR_TI_alloc(const IALG_Params *algParams, IALG_Fxns **pf,
                        IALG_MemRec memTab[]);

extern Int FIR_TI_free(IALG_Handle handle, IALG_MemRec memTab[]);

extern Int FIR_TI_initObj(IALG_Handle handle,
                          const IALG_MemRec memTab[], IALG_Handle parent,
                          const IALG_Params *algParams);

extern Void FIR_TI_moved(IALG_Handle handle,
                          const IALG_MemRec memTab[], IALG_Handle parent,
                          const IALG_Params *algParams);

extern Void FIR_TI_filter(IFIR_Handle handle, Int in[], Int out[]);

extern IRTC_Mask FIR_TI_rtcGet(IRTC_Handle handle);

extern Void FIR_TI_rtcBind(LOG_Obj *log);

extern Void FIR_TI_rtcSet(IRTC_Handle handle, IRTC_Mask mask);

#endif  /* FIR_TI_PRIV_ */
```

Name **fir_ti_ext.c – Vender specific FIR Extensions**

Text

```
/*
 *  ======== fir_ti_ext.c ========
 */
#pragma CODE_SECTION(FIR_TI_create, ".text:create")
#pragma CODE_SECTION(FIR_TI_delete, ".text:delete")
#pragma CODE_SECTION(FIR_TI_init, ".text:init")
#pragma CODE_SECTION(FIR_TI_exit, ".text:exit")

#include <std.h>
#include <alg.h>
#include <ialg.h>
#include <fir.h>
#include <ifir.h>

#include <fir_ti.h>
#include <fir_ti_priv.h>

/*
 *  ======== FIR_TI_create ========
 */
FIR_TI_Handle FIR_TI_create(const FIR_Params *params)
{
    return ((Void *)ALG_create(&FIR_TI_IALG,NULL,(IALG_Params *)params));
}

/*
 *  ======== FIR_TI_delete ========
 */
Void FIR_TI_delete(FIR_TI_Handle handle)
{
    ALG_delete((ALG_Handle)handle);
}

/*
 *  ======== FIR_TI_exit ========
 */
Void FIR_TI_exit(Void)
{
    ALG_exit();
}

/*
 *  ======== FIR_TI_init ========
 */
Void FIR_TI_init(Void)
{
    ALG_init();
}
```

```
/*
 *  ======== FIR_TI_nApply ========
 */
Void FIR_TI_nApply(FIR_TI_Handle handle, Int input[], Int n)
{
    Int *in;
    Int i;

    ALG_activate((ALG_Handle)handle);

    for (in = input, i = 0; i < n; i++) {
        FIR_TI_filter((IFIR_Handle)handle, in, in);
        in += handle->frameLen;
    }

    ALG_deactivate((ALG_Handle)handle);
}
```

Name **fir_ti_irtc.c – Vendor Specific Implementation of IRTC Interface**

Text

```
/*
 *  ======== fir_ti_irtc.c ========
 *  Filter Module IRTC implementation - TI's implementation of the
 *  IRTC interface for the FIR filter algorithm
 */
#include <std.h>

#include <irtc.h>
#include <fir_ti.h>
#include <fir_ti_priv.h>
#include <log.h>


/*
 *  ======== FIR_TI_rtcOut ========
 *  This module's output trace log.
 */
LOG_Obj *FIR_TI_rtcOut = NULL;

/*
 *  ======== FIR_TI_rtcBind ========
 */
Void FIR_TI_rtcBind(LOG_Obj *log)
{
    FIR_TI_rtcOut = log;

    FIR_TI_trace("FIR_TI_rtcBind(0x%lx)\n", log, NULL);
}


/*
 *  ======== FIR_TI_rtcGet ========
 */
IRTC_Mask FIR_TI_rtcGet(IRTC_Handle handle)
{
    FIR_TI_Obj *fir = (Void *)handle;

    FIR_TI_trace("FIR_TI_rtcGet(0x%lx) = 0x%x\n", handle, fir->mask);

    return (fir->mask);
}
```

```
/*
 *  ======== FIR_TI_rtcSet ========
 */
Void FIR_TI_rtcSet(IRTC_Handle handle, IRTC_Mask mask)
{
    FIR_TI_Obj *fir = (Void *)handle;

    FIR_TI_trace("FIR_TI_rtcSet(0x%lx, 0x%x)\n", handle, mask);

    fir->mask = mask;
}
```

Name

fir_ti_ifirvt.c – Vendor Specific IFIR Function Table

Text

```
/*
 *  ======== fir_ti_ifirvt.c ========
 *  This file contains the function table definitions for all
 *  interfaces implemented by the FIR_TI module that derive
 *  from IALG
 *
 *  We place these tables in a separate file for two reasons:
 *     1. We want to allow one to one to replace these tables
 *        with different definitions.  For example, one may
 *        want to build a system where the FIR is activated
 *        once and never deactivated, moved, or freed.
 *
 *     2. Eventually there will be a separate "system build"
 *        tool that builds these tables automatically
 *        and if it determines that only one implementation
 *        of an API exists, "short circuits" the vtable by
 *        linking calls directly to the algorithm's functions.
 */
#include <std.h>

#include <ialg.h>
#include <ifir.h>

#include <fir_ti.h>
#include <fir_ti_priv.h>

#define IALGFXNS \
    &FIR_TI_IALG,       /* module ID */                          \
    FIR_TI_activate,    /* activate */                           \
    FIR_TI_alloc,       /* alloc */                              \
    NULL,               /* control (NULL => no control ops) */   \
    FIR_TI_deactivate,  /* deactivate */                         \
    FIR_TI_free,        /* free */                               \
    FIR_TI_initObj,     /* init */                               \
    FIR_TI_moved,       /* moved */                              \
    NULL                /* numAlloc() (NULL => IALG_MAXMEMRECS) */\

/*
 *  ======== FIR_TI_IFIR ========
 *  This structure defines TI's implementation of the IFIR interface
 *  for the FIR_TI module.
 */
IFIR_Fxns FIR_TI_IFIR = {       /* module_vendor_interface */
    IALGFXNS,
    FIR_TI_filter    /* filter */
};
```

```
/*
 *  ======== FIR_TI_IALG ========
 *  This structure defines TI's implementation of the IALG interface
 *  for the FIR_TI module.
 */
#ifdef _TI_

asm("_FIR_TI_IALG .set _FIR_TI_IFIR");

#else


/*
 *  We duplicate the structure here to allow this code to be compiled and
 *  run non-DSP platforms at the expense of unnecessary data space
 *  consumed by the definition below.
 */
IALG_Fxns FIR_TI_IALG = {       /* module_vendor_interface */
    IALGFXNS
};

#endif
```

Name **fir_ti_irtcvt.c – Vendor Specific IRTC Function Table**

Text

```
/*
 *  ======== fir_ti_irtcvt.c ========
 *  This file contains the function table definitions for the
 *  IRTC interface implemented by the FIR_TI module.
 *
 *  We place these tables in a separate file for two reasons:
 *     1. We want allow one to one to replace these tables
 *        with different definitions.  For example, one may
 *        want to build a system where the FIR is activated
 *        once and never deactivated, moved, or freed.
 *
 *     2. Eventually there will be a separate "system build"
 *        tool that builds these tables automatically
 *        and if it determines that only one implementation
 *        of an API exists, "short circuits" the vtable by
 *        linking calls directly to the algorithm's functions.
 */
#include <std.h>

#include <irtc.h>

#include <fir_ti.h>
#include <fir_ti_priv.h>


/*
 *  ======== FIR_TI_IRTC ========
 *  This structure defines TI's implementation of the IRTC interface
 *  for the FIR_TI module.
 */
IRTC_Fxns FIR_TI_IRTC = {
    &FIR_TI_IALG,       /* module ID */
    FIR_TI_rtcBind,     /* rtcBind */
    FIR_TI_rtcGet,      /* rtcGet */
    FIR_TI_rtcSet       /* rtcSet */
};
```

Name

firtest.c – example client of FIR utility library

Text

```
/*
 *  ======== firtest.c ========
 *  This example shows how to use the type safe FIR "utility"
 *  library directly by an application.
 */
#include <std.h>
#include <fir.h>
#include <log.h>

#include <fir_ti.h>

#include <stdio.h>

extern LOG_Obj trace;

Int coeff[] = {1, 2, 3, 4, 4, 3, 2, 1};
Int input[] = {1, 0, 0, 0, 0, 0, 0};

#define FRAMELEN    (sizeof (input) / sizeof (Int))
#define FILTERLEN   (sizeof (coeff) / sizeof (Int))

Int output[FRAMELEN];

static Void display(Int a[], Int n);

/*
 *  ======== main ========
 */
Int main(Int argc, String argv[])
{
    FIR_Params firParams;
    FIR_Handle fir;

    FIR_init();

    firParams = FIR_PARAMS;
    firParams.filterLen = FILTERLEN;
    firParams.frameLen = FRAMELEN;
    firParams.coeffPtr = coeff;
    if ((fir = FIR_create(&FIR_TI_IFIR, &firParams)) != NULL) {
        FIR_apply(fir, input, output);      /* filter some data */
        display(output, FRAMELEN);          /* display the result */
        FIR_delete(fir);                    /* delete the filter */
    }
    FIR_exit();

    return (0);
}
```

```
/*
 *  ======== display ========
 */
static Void display(Int a[], Int n)
{
    Int i;

    for (i = 0; i < n; i++) {
        LOG_printf(&trace, "%d ", a[i]);
    }

    LOG_printf(&trace, "\n");
}
```

Name **firtest1.c – example client of ALG, RTC, and FIR**

Text

```
/*
 *  ======== firtest1.c ========
 *  This example shows how the trace interface (if implemented)
 *  can be used by an application.  It also shows how to create
 *  an algorithm instance object using the ALG interface.
 *
 *  The ALG interface allows one to create code that can create
 *  an instance of *any* XDAIS algorithm at the cost of a loss of
 *  type safety.
 */
#include <std.h>
#include <fir.h>
#include <alg.h>
#include <log.h>
#include <ialg.h>
#include <rtc.h>

#include <fir_ti.h>

extern LOG_Obj trace;

Int coeff[] = {1, 2, 3, 4, 4, 3, 2, 1};
Int input[] = {1, 0, 0, 0, 0, 0, 0};

#define FRAMELEN    (sizeof (input) / sizeof (Int))
#define FILTERLEN   (sizeof (coeff) / sizeof (Int))

Int output[FRAMELEN];

static Void display(Int a[], Int n);

/*
 *  ======== main ========
 */
Int main(Int argc, String argv[])
{
    FIR_Params firParams;
    ALG_Handle alg;
    RTC_Desc rtc;

    ALG_init();
    FIR_init();
    RTC_init();

    /* bind output log to FIR_TI module */
    RTC_bind(&FIR_TI_IRTC, &trace);

    /* create an instance of a FIR algorithm */
    firParams = FIR_PARAMS;
```

```
    firParams.filterLen = FILTERLEN;
    firParams.frameLen = FRAMELEN;
    firParams.coeffPtr = coeff;
    alg = ALG_create((IALG_Fxns *)&FIR_TI_IFIR, NULL,
                     (IALG_Params *)&firParams);

    /* if the instance creation succeeded, create a trace descriptor */
    if (alg != NULL && RTC_create(&rtc, alg, &FIR_TI_IRTC) != NULL) {

        RTC_set(&rtc, RTC_ENTER);                    /* enable trace */
        FIR_apply((FIR_Handle)alg, input, output);   /* filter data */
        display(output, FRAMELEN);                   /* display result */

        RTC_delete(&rtc);       /* delete rtc descriptor */
        ALG_delete(alg);        /* delete alg instance */
    }

    RTC_exit();
    FIR_exit();
    ALG_exit();
    return (0);
}

/*
 *  ======== display ========
 */
static Void display(Int a[], Int n)
{
    Int i;

    for (i = 0; i < n; i++) {
        LOG_printf(&trace, "%d ", a[i]);
    }

    LOG_printf(&trace, "\n");
}
```

Name

fig.h – Filter Group Module Interface

Text

```
/*
 *  ======== fig.h ========
 *  Filter Group Module Header - This module implements a FIR
 *  filter group object.  A filter group object simply
 *  maintains global state (common coefficients and working
 *  buffer) multiple FIR objects.  Thus, this module does not
 *  have a "process" method, it only implements "activate"
 * "deactivate", and "getStatus".
 */
#ifndef FIG_
#define FIG_

#include <ifig.h>

typedef struct IFIG_Obj *FIG_Handle;

/*
 *  ======== FIG_Params ========
 *  Filter group instance creation parameters
 */
typedef struct IFIG_Params FIG_Params;

extern const FIG_Params FIG_PARAMS; /* default instance parameters */

/*
 *  ======== FIG_Status ========
 *  Status structure for getting FIG instance attributes
 */
typedef struct IFIG_Status FIG_Status;

/*
 *  ======== FIG_activate ========
 */
extern Void FIG_activate(FIG_Handle handle);

/*
 *  ======== FIG_create ========
 */
extern FIG_Handle FIG_create(IFIG_Fxns *fxns, IFIG_Params *prms);

/*
 *  ======== FIG_deactivate ========
 */
extern Void FIG_deactivate(FIG_Handle handle);

/*
 *  ======== FIG_delete ========
 */
extern Void FIG_delete(FIG_Handle fir);
```

```
/*
 *  ======== FIG_getStatus ========
 */
extern Void FIG_getStatus(FIG_Handle fig, FIG_Status *status);

#endif  /* FIG_ */
```

Name

ifig.h – Example Abstract FIR Filter Group Interface

Text

```
/*
 *  ======== ifig.h ========
 *  Filter Group Module Header - This module implements a FIR filter
 *  group object.  A filter group object simply maintains global state
 *  (common coefficients and working buffer) multiple FIR objects.
 *  Thus, this module does not have a "process" method, it only
 *  implements "activate" and "deactivate".
 */
#ifndef IFIG_
#define IFIG_

#include <ialg.h>


/*
 *  ======== IFIG_Params ========
 *  Filter group instance creation parameters
 */
typedef struct IFIG_Params {
    Int size;           /* sizeof this structure */
    Int *coeffPtr;      /* pointer to coefficient array */
    Int filterLen;      /* length of coefficient array (words) */
} IFIG_Params;

extern const IFIG_Params IFIG_PARAMS;   /* default instance parameters */

/*
 *  ======== IFIG_Obj ========
 */
typedef struct IFIG_Obj {
    struct IFIG_Fxns *fxns;
} IFIG_Obj;

/*
 *  ======== IFIG_Handle ========
 */
typedef struct IFIG_Obj *IFIG_Handle;

/*
 *  ======== IFIG_Status ========
 *  Status structure for getting FIG instance attributes
 */
typedef struct IFIG_Status {
    Int *coeffPtr;          /* pointer to coefficient array */
} IFIG_Status;
```

```
/*
 *  ======== IFIG_Fxns ========
 */
typedef struct IFIG_Fxns {
    IALG_Fxns ialg;
    Void (*getStatus)(IFIG_Handle handle, IFIG_Status *status);
} IFIG_Fxns;

#endif  /* IFIG_ */
```

Name **fig.c – Common Filter Group Module Implementation**

Text

```
/*
 *  ======== fig.c ========
 *  Filter Group - this module implements a filter group; a group of FIR
 *  filters that share a common set of coefficients and a working buffer.
 */
#include <std.h>
#include <fig.h>

/*
 *  ======== FIG_exit ========
 */
Void FIG_exit(Void)
{
}

/*
 *  ======== FIG_init ========
 */
Void FIG_init(Void)
{
}
```

Name **fig_ti.c – Vendor-Specific Filter Group Implementation**

Text

```
/*
 *  ======== fig_ti.c ========
 *  Filter Group - this module implements a filter group; a group of FIR
 *  filters that share a common set of coefficients and a working buffer.
 */
#pragma CODE_SECTION(FIG_TI_alloc, ".text:algAlloc()")
#pragma CODE_SECTION(FIG_TI_free, ".text:algFree")
#pragma CODE_SECTION(FIG_TI_initObj, ".text:algInit")
#pragma CODE_SECTION(FIG_TI_moved, ".text:algMoved")

#include <std.h>
#include <ialg.h>
#include <fig_ti.h>
#include <ifig.h>
#include <string.h>     /* memcpy() declaration */

#define COEFF   1
#define NUMBUFS 2

typedef struct FIG_TI_Obj {
    IALG_Obj    alg;            /* MUST be first field of XDAIS algs */
    Int         *coeff;         /* on-chip persistant coefficient array */
    Int         filterLen;      /* filter length (in words) */
} FIG_TI_Obj;

/*
 *  ======== FIG_TI_alloc ========
 */
Int FIG_TI_alloc(const IALG_Params *algParams, IALG_Fxns **parentFxns,
                 IALG_MemRec memTab[])
{
    const IFIG_Params *params = (Void *)algParams;

    if (params == NULL) {
        params = &IFIG_PARAMS;  /* set default parameters */
    }

    /* Request memory for FIG object */
    memTab[0].size = sizeof (FIG_TI_Obj);
    memTab[0].alignment = 0;
    memTab[0].space = IALG_EXTERNAL;
    memTab[0].attrs = IALG_PERSIST;
```

```
    /*
     *  Request memory for filter coefficients
     *
     *  Note that this buffer is declared as persistent; i.e., it is the
     *  responsibility of the client to insure that its contents are
     *  preserved whenever this object is active.
     */
    memTab[COEFF].size = params->filterLen * sizeof(Int);
    memTab[COEFF].alignment = 0;
    memTab[COEFF].space = IALG_DARAM1;
    memTab[COEFF].attrs = IALG_PERSIST;

    return (NUMBUFS);
}

/*
 *  ======== FIG_TI_free ========
 */
Int FIG_TI_free(IALG_Handle handle, IALG_MemRec memTab[])
{
    FIG_TI_Obj *fig = (Void *)handle;
    FIG_TI_alloc(NULL, NULL, memTab);
    memTab[COEFF].base = fig->coeff;
    memTab[COEFF].size = fig->filterLen * sizeof (Int);

    return (NUMBUFS);
}

/*
 *  ======== FIG_TI_initObj ========
 */
Int FIG_TI_initObj(IALG_Handle handle,
                   const IALG_MemRec memTab[], IALG_Handle parent,
                   const IALG_Params *algParams)
{
    FIG_TI_Obj *fig = (Void *)handle;
    const IFIG_Params *params = (Void *)algParams;

    if (params == NULL) {
        params = &IFIG_PARAMS;  /* use defaults if algParams == NULL */
    }

    /* initialize the FIG object's fields */
    fig->coeff = memTab[COEFF].base;
    fig->filterLen = params->filterLen;

    /* copy coefficients into on-chip persistant memory */
    memcpy((Void *)fig->coeff,
        (Void *)params->coeffPtr, params->filterLen * sizeof (Int));

    return (IALG_EOK);
}
```

```
/*
 *  ======== FIG_TI_getStatus ========
 */
Void FIG_TI_getStatus(IFIG_Handle handle, IFIG_Status *status)
{
    FIG_TI_Obj *fig = (Void *)handle;
    status->coeffPtr = fig->coeff;
}

/*
 *  ======== FIG_TI_moved ========
 */
Void FIG_TI_moved(IALG_Handle handle,
                  const IALG_MemRec memTab[], IALG_Handle parent,
                  const IALG_Params *algParams)
{
    FIG_TI_Obj *fig = (Void *)handle;

    /* initialize the FIG object's fields */
    fig->coeff = memTab[COEFF].base;
}
```

Name **fig_ti.h – Vendor-Specific Filter Group Interface**

Text

```
/*
 *  ======== fig_ti.h ========
 *  Vendor specific (TI) interface header for Filter Group algorithm
 */
#ifndef FIG_TI_
#define FIG_TI_

#include <ialg.h>
#include <ifig.h>

/*
 *  ======== FIG_TI_exit ========
 *  Required module finalization function
 */
extern Void FIG_TI_exit(Void);

/*
 *  ======== FIG_TI_init ========
 *  Required module initialization function
 */
extern Void FIG_TI_init(Void);

/*
 *  ======== FIG_TI_IALG ========
 *  TI's implementation of FIG's IALG interface
 */
extern IALG_Fxns FIG_TI_IALG;

/*
 *  ======== FIG_TI_IFIG ========
 *  TI's implementation of FIG's IFIG interface
 */
extern IFIG_Fxns FIG_TI_IFIG;
#endif  /* FIG_TI_ */
```

Name **fig_ti_ifigvt.h – Vendor-Specific FIG Function Table**

Text

```
/*
 *  ======== fig_ti_ifigvt.c ========
 *  This file contains the function table definitions for all interfaces
 *  implemented by the FIG_TI module.
 */
#include <std.h>
#include <ialg.h>
#include <ifig.h>
#include <fig_ti.h>
#include <fig_ti_priv.h>

#define IALGFXNS \
    &FIG_TI_IALG,   /* implementation ID */                          \
    NULL,           /* activate (NULL => nothing to do) */           \
    FIG_TI_alloc,   /* alloc */                                      \
    NULL,           /* control (NULL => no control operations) */    \
    NULL,           /* deactivate (NULL => nothing to do) */         \
    FIG_TI_free,    /* free */                                       \
    FIG_TI_initObj, /* init */                                       \
    FIG_TI_moved,   /* moved */                                      \
    NULL            /* numAlloc() (NULL => IALG_MAXMEMRECS) */        \

/*
 *  ======== FIG_TI_IFIG ========
 */
IFIG_Fxns FIG_TI_IFIG = {       /* module_vendor_interface */
    IALGFXNS,               /* IALG functions */
    FIG_TI_getStatus        /* IFIG getStatus */
};

/*
 *  ======== FIG_TI_IALG ========
 *  This structure defines TI's implementation of the IALG interface
 *  for the FIG_TI module.
 */
#ifdef _TI_

asm("_FIG_TI_IALG .set _FIG_TI_IFIG");

#else
```

```
/*
 *  We duplicate the structure here to allow this code to be compiled and
 *  run non-DSP platforms at the expense of unnecessary data space
 *  consumed by the definition below.
 */
IALG_Fxns FIG_TI_IALG = {       /* module_vendor_interface */
    IALGFXNS,               /* IALG functions */
};

#endif
```

Name

fig_ti_priv.h – Private Vendor-Specific Filter Group Header

Text

```
/*
 *  ======== fig_ti_priv.h ========
 *  Internal vendor specific (TI) interface header for FIG
 *  algorithm. Only the implementation source files include
 *  this header; this header is not shipped as part of the
 *  algorithm.
 *
 *  This header contains declarations that are specific to
 *  this implementation and which do not need to be exposed
 *  in order for an application to use the FIG algorithm.
 */
#ifndef FIG_TI_PRIV
#define FIG_TI_PRIV

#include <ialg.h>

typedef struct FIG_TI_Obj {
    IALG_Obj    alg;            /* MUST be first field of XDAIS algs */
    Int         *coeff;         /* on-chip persistant coeffient array */
    Int         filterLen;      /* filter length (in words) */
} FIG_TI_Obj;

extern Int FIG_TI_alloc(const IALG_Params *,IALG_Fxns **, IALG_MemRec *);

extern Int FIG_TI_free(IALG_Handle, IALG_MemRec *);

extern Void FIG_TI_getStatus(IFIG_Handle handle, IFIG_Status *status);

extern Int FIG_TI_initObj(IALG_Handle,
                  const IALG_MemRec *, IALG_Handle, const IALG_Params *);

extern Void FIG_TI_moved(IALG_Handle,
                  const IALG_MemRec *, IALG_Handle, const IALG_Params *);

#endif
```

Name

figtest.c – Example Client of FIG and ALG

Text

```
/*
 *  ======== figtest.c ========
 *  Example use of FIG, FIR and ALG modules.  This test creates some
 *  number of FIR filters that all share a common set of coefficients
 *  and working buffer.  It then applies the filter to the data and
 *  displays the results.
 */
#include <std.h>
#include <fig.h>
#include <fir.h>
#include <log.h>

#include <fig_ti.h>
#include <fir_ti.h>

extern LOG_Obj trace;

#define NUMFRAMES   2           /* number of frames of data to process */

#define NUMINST     4           /* number of FIR filters to create */
#define FRAMELEN    7           /* length of in/out frames (words) */
#define FILTERLEN   8           /* length of coeff array (words) */

Int coeff[FILTERLEN] = {        /* filter coefficients */
    1, 2, 3, 4, 4, 3, 2, 1
};

Int in[NUMINST][FRAMELEN] = {   /* input data frames */
    {1, 0, 0, 0, 0, 0, 0},
    {0, 1, 0, 0, 0, 0, 0},
    {0, 0, 1, 0, 0, 0, 0},
    {0, 0, 0, 1, 0, 0, 0}
};
Int out[NUMINST][FRAMELEN];     /* output data frames */

static Void display(Int a[], Int n);

/*
 *  ======== main ========
 */
Int main(Int argc, String argv[])
{
    FIG_Params figParams;
    FIR_Params firParams;
    FIG_Status figStatus;
    FIG_Handle group;
    FIR_Handle inst[NUMINST];
    Bool status;
```

```
Int i, n;

FIG_init();
FIR_init();

figParams = FIG_PARAMS;
figParams.filterLen = FILTERLEN;
figParams.coeffPtr = coeff;

/* create the filter group */
if ((group = FIG_create(&FIG_TI_IFIG, &figParams)) != NULL) {

    /* get FIG pointers */
    FIG_getStatus(group, &figStatus);

    /* create multiple filter instance objects that reference group */
    firParams = FIR_PARAMS;
    firParams.frameLen = FRAMELEN;
    firParams.filterLen = FILTERLEN;
    firParams.coeffPtr = figStatus.coeffPtr;
    for (status = TRUE, i = 0; i < NUMINST; i++) {
        inst[i] = FIR_create(&FIR_TI_IFIR, &firParams);
        if (inst[i] == NULL) {
            status = FALSE;
        }
    }
    /* if object creation succeeded, apply filters to data */
    if (status) {
        /* activate group object */
        FIG_activate(group);

        /* apply all filters on all frames */
        for (n = 0; n < NUMFRAMES; n++) {
            for (i = 0; i < NUMINST; i++) {
                FIR_apply(inst[i], in[i], out[i]);
                display(out[i], FRAMELEN);
            }
        }
        /* deactivate group object */
        FIG_deactivate(group);
    }

    /* delete filter instances */
    for (i = 0; i < NUMINST; i++) {
        FIR_delete(inst[i]);
    }

    /* delete filter group object */
    FIG_delete(group);
}
FIG_exit();
FIR_exit();
```

```
    return (0);
}

/*
 *  ======== display ========
 */
static Void display(Int a[], Int n)
{
    Int i;

    for (i = 0; i < n; i++) {
        LOG_printf(&trace, "%d ", a[i]);
    }

    LOG_printf(&trace, "\n");
}
```